Interview Summary	**Application No.** 10/677,081	**Applicant(s)** HERSCHLEB, GARRETT	
	Examiner Tuan A. Vu	**Art Unit** 2193	

All participants (applicant, applicant's representative, PTO personnel):

(1) *Tuan A. Vu*. (3)_____.

(2) *Gregg Peacodk*. (4)_____.

Date of Interview: *23 October 2007*.

Type: a)☒ Telephonic b)☐ Video Conference
c)☐ Personal [copy given to: 1)☐ applicant 2)☐ applicant's representative]

Exhibit shown or demonstration conducted: d)☐ Yes e)☐ No.
If Yes, brief description: _____.

Claim(s) discussed: *1,2,7,15,19,22,23 and 27*.

Identification of prior art discussed: *n/a*.

Agreement with respect to the claims f)☒ was reached. g)☐ was not reached. h)☐ N/A.

Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: *See Continuation Sheet*.

(A fuller description, if necessary, and a copy of the amendments which the examiner agreed would render the claims allowable, if available, must be attached. Also, where no copy of the amendments that would render the claims allowable is available, a summary thereof must be attached.)

THE FORMAL WRITTEN REPLY TO THE LAST OFFICE ACTION MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a reply to the last Office action has already been filed, APPLICANT IS GIVEN A NON-EXTENDABLE PERIOD OF THE LONGER OF ONE MONTH OR THIRTY DAYS FROM THIS INTERVIEW DATE, OR THE MAILING DATE OF THIS INTERVIEW SUMMARY FORM, WHICHEVER IS LATER, TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW. See Summary of Record of Interview requirements on reverse side or on attached sheet.

Examiner Note: You must sign this form unless it is an Attachment to a signed Office action.

Examiner's signature, if required

Continuation of Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: the substance of the communication was about propagating some limitation from claim 1 to the rest of the independent claims while maintaining statutory status of the claimed steps in a proper sequence that would yield a application result, as set forth in the intended purpose of accessing stack parameter by implicit reference passing. The examiner and the representative (during 10/22-10/25) worked out a common scenario to set forth the 3 executing steps with emphasis on the offset usage, and as agreed, the amendments when resubmitted would put the case in condition for an Examiner's amendments.

S/N 10/677,081 **PATENT**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Garrett Herschleb	Examiner: Tuan Vu
Serial No.:	10/677,081	Group Art Unit: 2193
Filed:	September 30, 2003	Docket: 884.A78US1
Title:	PASSING PARAMETERS BY IMPLICIT REFERENCE	

EXAMINER INTERVIEW SUMMARY

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

The Applicant thanks Examiner Tuan Vu for the telephone interview granted on October 23, 2007, between himself and the Applicant's representative Mr. Gregg Peacock (Reg. No. 45,001). Applicant's representative granted authorization to amend claims 1, 7, 15, 19, 22, and 24-30 and cancel claims 2, 16, and 23.

Please amend the above-identified application as follows:

CLAIMS

Please amend the claims as follows:

1. (Currently Amended) A method comprising:

compiling a program code to generate an executable program that comprises accessing a parameter on a stack using an implicit reference, wherein executing the executable program comprises:

~~executing an executable program that comprises:~~

executing a first module that includes a first instruction that causes a variable to be loaded onto ~~a~~ the stack, wherein the first module includes a second instruction that calls a second module that passes ~~a~~ the parameter by the implicit reference, wherein the parameter passed by reference is an address of the variable; and

executing the second module that includes a third instruction that accesses the parameter on the stack based on an offset of a base pointer of ~~from a common reference for~~ the second module.

2. (Canceled)

3. (Original) The method of claim 1, wherein the parameter is not pushed onto the stack as part of the call from the first module to the second module.

4. (Original) The method of claim 1, wherein executing the first module that includes the first instruction that causes the variable to be loaded onto the stack comprises executing the first module that includes the first instruction to define the variable.

5. (Original) The method of claim 1, wherein the first module calls the second module through a third module, wherein a first calling sequence comprises the first module, the third module and the second module.

6. (Original) The method of claim 5, wherein a second calling sequence comprises the first module, a fourth module and the second module, wherein executing the first module comprises padding the stack such that the number of entries on the call stack are the same for the first calling sequence and the second calling sequence.

7. (Currently Amended) A method comprising:

receiving a program code that includes a first calling function having a call instruction to call a first called function that passes, by reference, a variable as a parameter of a number of parameters; and

compiling the program code to generate compiled instructions, wherein the compiled instructions do not include a compiled instruction to load the variable onto a call stack, during execution, based on the call instruction; and

executing the compiled instructions that comprises accessing the parameter on the call stack using an implicit reference, wherein executing the compiled instructions comprises,

executing a first module that includes a first instruction that causes the variable to be loaded onto the call stack, wherein the first module includes a second instruction that calls a second module that passes the parameter by the implicit reference, wherein the parameter passed by reference is an address of the variable; and

executing the second module that includes invoking a third instruction that accesses the parameter on the stack based on an offset of a base pointer of the second module.

8. (Original) The method of claim 7, wherein the first calling function has a declaration instruction to allocate the variable, wherein the compiled instructions are to cause the variable to be loaded onto the call stack, during execution, based on the declaration instruction.

9. (Original) The method of claim 8, wherein the called function includes an access instruction, wherein the compiled instructions are to cause the variable to be accessed from the call stack, during execution, based on the access instruction.

10. (Original) The method of claim 9, wherein the compiled instructions are to cause, during execution, the variable to be accessed using a base pointer associated with the called function.

11. (Original) The method of claim 9, wherein the first calling function calls the called function by calling a separate function, wherein the separate function calls the called function.

12. (Original) The method of claim 11, wherein a first calling sequence comprises the first calling function, the separate function and the called function and wherein the first calling function calls the called function through a second calling sequence, the compiled instructions having a compiled instruction to pad the call stack such that the number of entries on the call stack are the same for the first calling sequence and the second calling sequence.

13. (Original) The method of claim 7, wherein receiving program code comprises receiving program code that includes a second calling function to call the first called function that passes, by reference, the variable as the parameter of the number of parameters, wherein an order of the number of parameters passed by the first calling function equals an order of the number of parameters passed by the second calling function.

14. (Original) The method of claim 7, wherein receiving the program code comprises receiving the program code that includes a second calling function, wherein the compiled instructions are to cause an order of the number of parameters on the call stack for the first calling function include parameters for the first called function and parameters for a second called function and wherein an order of the number of parameters on the call stack for the second calling function include parameters for the first called function and parameters for a third called function, wherein a size of the parameters for the third called function are not greater than a size of the parameters for the second called function.

15. (Currently Amended) An apparatus comprising:

a processor; and

a compiler to execute in the processor to generate assembly code instructions, wherein the assembly code instructions comprise a first procedure and a second procedure, wherein the first procedure includes a first assembly code instruction to cause a variable to be loaded onto a call stack, wherein the first procedure includes a second assembly code instruction to cause the second procedure to be called, which includes passage of a parameter by reference, wherein the parameter passed by reference is an address of the variable, the second procedure to include a third assembly code instruction that is to access the parameter on the stack based on an offset of a base pointer of ~~from a common reference associated with~~ the second procedure.

16. (Canceled)

17. (Original) The apparatus of claim 15, wherein the assembly code instructions do not include an assembly code instruction to push the parameter onto the call stack as part of the call from the first procedure to the second procedure.

18. (Original) The apparatus of claim 15, wherein the first assembly code instruction is to define the variable.

19. (Currently Amended) A system comprising:

a secondary storage to store at least a part of a source code program, wherein the source code program includes a first subroutine that includes a first instruction to define a variable and a second instruction to call a second subroutine, wherein the call to the second subroutine includes passage of the variable by reference;

a dynamic random access memory to store at least a part of a call stack; and

a processor to compile the source code program to generate assembly code instructions, wherein the assembly code instructions do not cause the variable to be pushed onto the call stack based on the second instruction, wherein the processor is to generate an executable program from the assembly code instructions, the processor to execute the executable program that comprises access of the variable on the call stack using an implicit reference, wherein execution of the executable program comprises execution of a first module that includes a first instruction that causes the variable to be loaded onto the call stack, wherein the first module includes a second instruction that calls a second module that passes a parameter by the implicit reference, wherein the parameter passed by reference is an address of the variable, and wherein execution of the executable program comprises execution of the second module that includes invoking a third instruction that accesses the parameter on the stack based on an offset of a base pointer of the second module.

20. (Original) The system of claim 19, wherein at least one of the assembly code instructions is to cause the variable to be pushed onto the call stack based on the first instruction.

21. (Original) The system of claim 19, wherein the second subroutine includes a third instruction that is to access the variable, wherein at least one of the assembly code instructions is to cause access of the variable from the call stack.

22. (Currently Amended) A machine-readable storage medium that provides instructions, which when executed by a machine, cause said machine to perform operations comprising:

executing an executable program that comprises accessing a parameter on a stack using an implicit reference, wherein executing the executable program comprises,~~:~~

executing a first module that includes a first instruction that causes a variable to be loaded onto a stack, wherein the first module includes a second instruction that calls a second module that passes ~~a~~ the parameter by the implicit reference, wherein the parameter passed by the implicit reference is an address of the variable; and

executing the second module that includes a third instruction that accesses the parameter on the stack based on an offset of a base pointer of ~~from a common reference for~~ the second module.

23. (Canceled)

24. (Currently Amended) The machine-readable storage medium of claim 22, wherein the parameter is not pushed onto the stack as part of the call from the first module to the second module.

25. (Currently Amended) The machine-readable storage medium of claim 22, wherein the first module calls the second module through a third module, wherein a first calling sequence comprises the first module, the third module and the second module.

26. (Currently Amended) The machine-readable storage medium of claim 25, wherein a second calling sequence comprises the first module, a fourth module and the second module, wherein executing the first module comprises padding the stack such that the number of entries on the call stack are the same for the first calling sequence and the second calling sequence.

27. (Currently Amended) A machine-readable storage medium that provides instructions, which when executed by a machine, cause said machine to perform operations comprising:

receiving a program code that includes a calling function having a call instruction to call a called function that passes, by reference, a variable as a parameter; and

compiling the program code to generate compiled instructions, wherein the compiled instructions do not include a compiled instruction to load the variable onto a call stack, during execution, based on the call instruction; and

executing the compiled instructions that comprises accessing the parameter on the call stack using an implicit reference, wherein executing the compiled instructions comprises,

executing a first module that includes a first instruction that causes the variable to be loaded onto the call stack, wherein the first module includes a second instruction that calls a second module that passes the parameter by the implicit reference, wherein the parameter passed by the implicit reference is an address of the variable; and

executing the second module that includes a third instruction that accesses the parameter on the stack based on an offset of a base pointer of the second module.

28. (Currently Amended) The machine-readable storage medium of claim 27, wherein the calling function has a declaration instruction to allocate the variable, wherein the compiled instructions are to cause the variable to be loaded onto the call stack, during execution, based on the declaration instruction.

29. (Currently Amended) The machine-readable storage medium of claim 28, wherein the called function includes an access instruction, wherein the compiled instructions are to cause the variable to be accessed from the call stack, during execution, based on the access instruction.

30. (Currently Amended) The machine-readable storage medium of claim 29, wherein the compiled instructions are to cause, during execution, the variable to be accessed using a base pointer associated with the called function.

Conclusion

Applicant respectfully submits that the claims are in condition for allowance and notification to that effect is earnestly requested. The Examiner is invited to telephone Applicant's attorney at (612) 371-2103 to facilitate prosecution of this application.

If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 19-0743.

Respectfully submitted,

SCHWEGMAN, LUNDBERG & WOESSNER, P.A.
P.O. Box 2938
Minneapolis, MN 55402
(612) 371-2103

Date 10-25-07 By

Gregg A. Peacock
Reg. No. 45,001

I hereby certify that this paper is being transmitted by facsimile to the U.S. Patent and Trademark Office on the date shown below.

Amy J. Moriarty

Date of Transmission